Attorney Docket: 042390.P12310

APPLICATION FOR UNITED STATES PATENT

FOR

METHOD FOR INDICATING COMPLETION STATUS OF ASYNCHRONOUS EVENTS

Inventor(s): Linden Minnick
Roy Callum
Patrick L. Connor

Prepared by: Blakely, Sokoloff, Taylor & Zafman LLP
12400 Wilshire Boulevard, 7th Floor
Los Angeles, California 90025
Phone: (206) 292-8600
Facsimile: (206) 292-8606

CERTIFICATE OF MAILING via EXPRESS MAIL

I hereby certify that this paper or fee is being deposited with the United States Postal Service "Express Mail Post Office to Addressee" service under 37 CFR § 1.10 on the date indicated above and addressed to the Assistant Commissioner for Patents, Box Patent Application, Washington, D.C. 20231.

"Express Mail" Label Number EL862051389US — Date of Deposit November 15, 2001

Rimma N. Oks — Date 11-15-01

METHOD FOR INDICATING COMPLETION STATUS OF ASYNCHRONOUS EVENTS

TECHNICAL FIELD

This disclosure relates generally to controllers, and in particular, but not exclusively, relates to indicating completion status for asynchronous events.

BACKGROUND

Various types of device controllers, such as input/output (I/O) controllers, complete tasks assigned to them by a driver in the order the commands for the tasks are issued. For example, if an I/O controller is asked to send a data packet A, then a data packet B and then a data packet C, it will send the packets in that exact order. For simple tasks such as transmitting packets, or for tasks that are roughly equal in complexity and execution time, this serial operation is desirable. But in more complex operations where a controller is simultaneously working on several tasks in parallel, completion of tasks in the order of command issuance may not be desirable, such as where the tasks to be performed vary widely in completion time. For example, in a case where a controller is instructed to (1) encrypt a large block of data, and (2) transmit an unrelated packet of data, the packet transmission operation may use different functional units within the controller than the encryption operation. In this case, the transmit operation can be completed well before the large block encryption. But with the prior art operational-descriptor interfaces, the commands must be completed in order of issuance. Therefore, even if a controller is capable of out-of-order command completion, there is no method for the controller to signal command completion in any order different than the order in which the commands were issued.

Forcing the controller to complete the commands in order has serious implications for the performance of a device in which the controller is installed. For controllers that can accomplish a combination of complex and simple commands, the complex commands may add significant undue latency to the

completion indication of the subsequent simple commands. Certain computer applications are also hindered, for example when the application needs a completion indication before proceeding to some subsequent task.

One approach that has been tried is simply to add multiple operation-descriptor interfaces (*i.e.*, command interfaces or descriptor rings) to the device. The primary drawback of this approach is the cost associated with each additional command interface. Another complexity results from the mechanism that may be used to implement additional functions (such as encryption and authentication) within a network controller. As the latency for completing an event increases, it becomes more complex for the controller to keep track of the order of the original commands.

BRIEF DESCRIPTION OF THE DRAWINGS

Non-limiting and non-exhaustive embodiments of the present invention are described herein with reference to the following figures, wherein like reference numerals refer to like parts throughout the various views unless otherwise specified.

Figure 1A is a drawing of an example of a controller issuing commands and receiving the completion status for each command.

Figure 1B is a drawing illustrating the descriptor ring concept by which the controller of Figure 1A received completion notifications.

Figure 2 is a block drawing of a computing platform including an embodiment of the present invention.

Figure 3 is a drawing of a device that embodies the present invention.

Figure 4 is a block drawing of a first embodiment of the present invention.

Figure 5 is a block drawing of a second embodiment of the present invention.

DETAILED DESCRIPTION OF THE ILLUSTRATED EMBODIMENTS

Embodiments of a system and method for indicating completion status for asynchronous events are described herein. In the following description, numerous specific details are described to provide a thorough understanding of embodiments of the invention. One skilled in the relevant art will recognize, however, that the invention can be practiced without one or more of the specific details, or with other methods, components, materials, etc. In other instances, well-known structures, materials, or operations are not shown or described in detail to avoid obscuring aspects of the invention.

Reference throughout this specification to "one embodiment" or "an embodiment" means that a particular feature, structure, or characteristic described in connection with the embodiment is included in at least one embodiment of the present invention. Thus, appearances of the phrases "in one embodiment" or "in an embodiment" do not necessarily refer to the same embodiment. Furthermore, the particular features, structures, or characteristics may be combined in any suitable manner in one or more embodiments.

Figures 1A and 1B together show an example in which a controller 10 issues a plurality of commands 1-5 in a specified order to a computational resource 12 which will execute the commands. Generally, each command is part of an operational descriptor which, aside from the command itself, includes a "done indication" 14 whose value depends on whether the command contained in the operational descriptor has been completed. One example of a "done indication" is a "done bit," which may have a value of 0 until the command is completed, and a value of 1 after completion. Each operational descriptor is stored in a memory location on the controller, and the operational descriptor remains in the same memory location until the "done indication" indicates that the command has been

completed. When a command has been completed, a new operational descriptor can then overwrite the memory location taken up by the completed command.

After the controller issues the commands, the controller is notified of the completion of each command in the same order in which the commands were issued. The controller 10 thus implicitly assumes that, if notice has not been received that a task has been completed, then subsequent tasks in the order cannot have been completed; this is true even if the commands themselves were executed out of order. Thus, if the commands were issued in the order shown (1-5), the completion notification for each command must also come back in that order. Thus, in many systems the commands are issued in order, but a later command in the queue must wait for receipt of completion notification of all earlier-issued commands before the later command can be completed.

Figure 1B illustrates one example of a "descriptor ring" concept that illustrates how both the commands and their completion notifications are done serially. Both the commands and their completion notification must follow the path of the ring 16. Thus, the controller issues command 1, then proceeds along the ring 16 to issue command 2, then command 3, then command 4, and so forth. Proceeding around the ring 16, when all the commands have been issued it is time to receive the completion notification for each command. Again starting on the ring 16 at number 1, completion notification for command 1 is received, and then completion notification for command 2 is received, then further along the ring, where completion notification of command 3 is received, and so forth. In other examples of descriptor rings, all commands need not be issued before any completion indications are received. In one example, commands and their completion notifications could be alternately issued and received, such that command 1 could be issued, then completion notification 1 received, then command 2 issued, then completion notification 2 received, and so on. Regardless of the particular sequencing of commands and their completion notifications, the

distinguishing feature of a descriptor ring is that a subsequent command must wait for completion notification of a previous command. This descriptor ring concept of serial issuance of commands and serial receipt of completion notification has various disadvantages. For example, some applications will not send a new request until the previous one has been completed. The current scheme, which requires the notification of completion in the same order as the commands were issued thus increases latency in applications because it slows down notification of completion.

Figure 2 illustrates the components of an embodiment of a computing platform 60 utilizing an embodiment of the present invention. The computing platform 60 will typically be a computing device such as a personal computer (PC), server, or the like. The platform 60 includes a processor 62, a memory 64, a user interface 68, a platform bus 70, drivers 72, and a multifunction controller 20. The processor, memory, user interface, drivers and multifunction controller are all connect to, and in communication with the platform bus 70. The platform bus 70 functions as a system-wide communication channel, so that by virtue of their connection to the platform bus 70, the other components are in communication with each other as well. Also attached to the processor 62 is a cache memory 74, which provides a certain amount of local, high-speed memory to the processor. An embodiment of the multifunction controller 20 is further discussed below.

Figure 3 shows a device in the form of a multifunction controller 20 capable of communication and cryptography operations that illustrates an embodiment of the present invention, which allows a completion status to be written to a programmable location, rather than to the same memory location where the operational descriptor including the command is stored. Although discussed herein in the context of a multifunction controller 20, the device could be of another type, and could be implemented differently, such as by implementing all the functions of the multifunction controller on a single chip. The multifunction controller 20 includes a plurality of units that are integrated into the multifunction

controller 20. The plurality of units in the multifunction controller 20 includes a bus interface controller 24, an input-output (I/O) unit 26 and various computational units that carry out commands issued by the driver. In this embodiment, the computational resources include a first encryption unit 28, a second encryption unit 30, and a third encryption unit 32. The first, second and third encryption units 28, 30 and 32 on the multifunction controller 20 each implements a different type of encryption algorithm, depending on the level of security a user wants. For example, the first unit 28 may implement an AES encryption engine, the second unit 30 may implement a 3DES encryption engine, and the third unit 32 may implement a third encryption scheme. Data Encryption Standard (DES) is a block cipher algorithm for encrypting data; it both enciphers and deciphers data using a 40-bit or 56-bit key specified in the Federal Information Standard publication 46, dated January 15, 1977. Advanced Encryption Standard (AES) is a standard for encryption intended to replace DES. AES is a symmetric, or private key, algorithm with a block cipher supporting key length ranging from 128 to 256 bits and variable length blocks of data.

In operation, the multifunction controller 20 receives input, such as commands from the driver software, through bus interface controller 24, and in response it issues commands to the various computational units on the multifunction controller 20 to achieve the desired result. The controller is in communication with a memory 36, or alternatively with the memory 64 of the computing platform 60 in which the multifunction controller 20 is installed. Among other functions, the memory stores operational descriptors containing commands that are to be issued or have been issued, and also receives the notification completion indication when each of the commands have been executed. If a user wants only a low level of security, they instruct their application to encrypt using the first encryption unit 28. The driver then sends the data to the multifunction controller 20multifunction controller, where the first encryption unit

28, encrypts the data and signals a notification that the task has been completed. When the driver receives the completion notification from the encryption unit 28, the driver then issues an instruction to the encryption unit to transfer the data to the I/O unit 26. Once the data transfer is complete, the controller might instruct the I/O unit 26 to send the data to another destination, such as another card in a communication system, or out over a computer network. If the user wants a higher level of security, they instruct their application to use one of the other encryption units 30 or 32, and the process is similar. When the controller 24 issues commands to the different components on the card, the components each notify the controller when the command has been completed. Alternatively, a user may want to simultaneously perform two independent operations, such as an encryption command executed by one of the encryption units 28, 30 or 32, and a transmit command executed by the I/O unit 26. Since these operations are independent the multifunction controller can perform them in any order, although to reduce latency in communications it is desirable to execute the transmit command first because it has a much shorter execution time than the encryption operation.

Of course, the multifunction controller 20 is only one possible embodiment of a device that can be used in the present invention; other embodiments that carry out different functions and include different components (*e.g.*, components other than encryption units) are possible. Different numbers of computational resources on the device are also possible.

Figure 4 illustrates an embodiment of how computational resources on the multifunction controller 20 can communicate completion of commands issued by the driver. In this embodiment, a controller 40 has a plurality of memory locations 42 in which it stores commands to be issued. Each command is contained in an operational descriptor which includes, among other things, the command itself, a value to be written upon completion of the command, and a memory address which points to or identifies a memory location to which the command's

completion indication is to be written. The memory address in the operational descriptor can either be an absolute address or, if memory conservation is necessary, can be an offset from a base memory address. The value to be written upon completion may be any value, including for example the original memory location of the command, an indication that it is a command that must be completed in a specific sequence, and so forth.

In operation, the controller issues the commands in the order they are stacked in the memory locations 42. The commands are categorized, for example according to which component they use on the card 20. Thus, the "A" commands A1 and A2 can correspond, in the multifunction controller 20, to commands issued to the first encryption unit 28; the "B" command B1 to the second encryption unit 30; and the "C" commands C1 and C2 to the third encryption unit 32. Other categorizations are also possible. For example, in another embodiment where there is only one computational resource to execute the commands, the commands might be categorized according to their execution time. In such an embodiment, the "A" commands could be those with small execution times, the "B" commands could be those with medium execution times, and the "C" commands could be those with long execution times.

Once the commands are issued by the controller 40, the appropriate computational resource of the multifunction controller executes the commands sent to it. As each command is completed, its completion status is written to the memory location included in the operational descriptor that contained the command, which will usually be a different location from the memory address where the command originated. In the embodiment shown, three blocks of memory addresses are used to write the completion status: one block 44 for the "A" commands, a block 46 for the "B" commands, and a block 48 for the "C" commands. For each category of commands, the memory blocks have several memory addresses corresponding to the number of commands in that category. As

described above, the operational descriptor for each command may either contain an absolute memory address or an offset from a base address. In this embodiment, for example, each of the memory blocks can start at a base address, and the operational descriptor for a command in the category corresponding to that block will contain an offset from the base address for that block. The memory blocks 44, 46 and 48 can be allocated prior to execution of the commands, or may be allocated dynamically according to how many commands there are, or according to the number of categories into which the commands are grouped.

Because the completion status of each command is now written to an arbitrarily programmed memory location, the mechanism for accepting and executing commands is decoupled from the mechanism for reporting their completion, meaning that the component executing the command can write its completion status to that location as soon as it is finished. In contrast to the "descriptor ring" structure shown in the prior art, in the embodiment described the resource need not wait for completion indication of commands issued prior to the one just executed before writing its completion status. Of course, there are situations in which maintaining a certain order of execution and completion of the commands is necessary. For example, commands that execute transmission or reception of data through the I/O unit 26 (Figure 1) to a computer network must send all the packets in the proper order. To accomplish this, certain commands could be set so that they must be executed in order, for example by including an order indicator in the operational descriptor.

Figure 5 illustrates an alternative or additional embodiment of the invention. As in the embodiment shown in Figure 4, this embodiment includes a controller 50 having memory locations 52 in which it stores operational description containing commands to be issued. As shown, the commands are commands A1, B1, A2, and so forth. The controller issues the commands to the appropriate computational resource in the order in which they are stacked in the memory

locations 52. As before, the commands are categorized, for example according to which computational resource they use, or according to the time required to execute them. In this embodiment, however, every command in a given category contains the same memory address in the operational descriptor.

As each command is completed, its completion status is written to the memory location contained in its operational descriptor, which will usually be a different location from the memory location 52 where the commands are stacked. In this embodiment, three memory locations are used to write the completion status: one location 54 for the "A" commands, an location 56 for the "B" commands, and a location 58 for the "C" commands. Because each category now has a single memory address to which the completion status is written, a completion status of a command simply overwrites the completion status of an earlier command. This embodiment would be useful to save memory where, for example, all commands in a category must be performed in a specific order, so that the completion value of a later command can overwrite the completion value of an earlier command without creating any difficulties. The memory addresses 54, 56 and 58 can be allocated prior to execution of the commands, or may be allocated dynamically according to the number of categories into which the commands are grouped. The memory addresses 54, 56, and 58 can also correspond to different categories, such as command execution time, as discussed above.

The above description of illustrated embodiments of the invention, including what is described in the Abstract, is not intended to be exhaustive or to limit the invention to the precise forms disclosed. While specific embodiments and examples of the invention are described herein for illustrative purposes, many equivalent modifications are possible within the scope of the claims, as those skilled in the relevant art will recognize. These modifications can be made to the invention in view of the above detailed description.

The terms used in the following claims should not be construed to limit the invention to the specific embodiments disclosed in the specification and the claims. Rather, the scope of the invention is to be determined entirely by the following claims, which are to be construed in accordance with established doctrines of claim interpretation.